Plan to implement SMB 3 SMB-Direct support in Samba
===================================================================
(State: 30.1.2020)
===================

The plan is split into 3 parts:
1. Tasks required to add SMB 3 SMB-Direct support:
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Lay out what is required in order to develop SMB-Direct with Samba on top of a recent Linux Kernel. The result is a setup that can be used in production, without bringing the changes into the upstream master branches of Samba and Linux.
   
   => 10-14w (~2.5-3.5M)

2. Tasks required to bring SMB 3 SMB-Direct to Upstream:
   ~~~~~~~~~~~~~~~~~~~~~~
   Builds on top of section 1. and includes the work for automated regression tests and discussions/adjustments in order to get the changes accepted by the upstream Samba and Linux communities.

   => 6-10w (~1.5-2.5M)

3. Development Requirements:
   ~~~~~~~~~~~~~~~~~~
   This plan assumes running on a Linux host.

1. Tasks required to add SMB 3 SMB-Direct support
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   This is just the first part that is really required in order to get something that can be used, e.g. it speaks the required protocols and it build on top of recent Samba and Linux-Kernel version (at the time the project finishes).

   The feature development will be designed with upstream inclusion into Samba and the Linux-Kernel in mind.

   But the required effort for upstream inclusion can be found in section 2.

1.1 Multi-Channel
--------------

In order to provide the SMB-Direct feature to Windows clients, it's required to finish the Multi-Channel feature in Samba.

While a lot of things are already in current Samba releases, there are still things to do in order to get this into a useable state some, which could be considered production ready.

1.1.1 Multi-Channel Server-Client retry
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   - enable keepalives and TCP_USER_TIMEOUT (on Linux)
   - Functions like getsockopt(TCP_INFO), ioctl(SIOCINQ), ioctl(fd, SIOCOUTQ) or ioctl(SIOCOUTQNSD) are hopefully able to detect if some bytes are already (tcp) acked by the client. On linux we can simply use TCP_USER_TIMEOUT.
   - retry to send Oplock/Lease Breaks on a different channel (if there is more than one channel)
   - For testing we would have a special fsctl call specifying a timer when to trigger the disconnect of a channel. Only allowed with on option similar to "smbd:suicide mode".
   - See https://bugzilla.samba.org/show_bug.cgi?id=11897
   - See https://github.com/spuiuk/samba.git
     github-spuiuk/sp-multichannel.20190618
   - TESTS
   - TODO: ask dochelp:
     - is it ok for Windows client to get multiple oplock/lease breaks
   
   => 1-2w
1.1.2 Interface discovery in a cluster
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

scope:
~~~~~~
- The interaction with ctdd cluster
- multiple channels on a single node, not on multiple nodes simultaneously
  (this matches the behaviour of Windows SOFS clusters)
- in a ctdd cluster, we should make sure that we only
  return ip addresses local to the node.
  possibly ctdd needs change / be configured to
  not handle public addresses.
- one client will only ever connect to one cluster node.

tasks:
~~~~~~
- The implementation of FSCTL_QUERY_NETWORK_INTERFACE_INFO
  should use CTDB_CONTROL_GET_PUBLIC_IPS and blacklist
  all public addresses from the returned response.
  => 0.5w

1.1.3 Session teardown on ip movement
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

- If smbd gets a CTDB_SRVID_RELEASE_IP message
  from ctdd for an ip that's attached to a
  ctdd public address, it should shutdown all
  connected channels instead of just the one
  attached to the specific ip address.
  This means the client needs to reconnect completely
  (most likely to another node) and will ask for
  available MultiChannel interfaces again.
  => 0.5w

  => 1-3w (~0.25-0.75M) (1.1 Multi-Channel)

1.2 Linux Kernel Driver
--------------------------

There're userspace libraries (libibverbs and librdmacm), which offer support
for RDMA communication, while bypassing the kernel. But these libraries don't
support fd-passing, which is used by Samba in order to support Multi-Channel.

There was a userspace SMB-Direct proxy daemon, but it's performance is very
bad.

The current design consists of a Kernel driver that provides a socket interface
with some additional features for RDMA data transfers.

The SMB3 client in the Linux Kernel has experimental support for SMB-Direct
(since 4.15). The new SMB-Direct driver can also be used there, which might help
to get it merged upstream.

There's currently work in progress for
ibv_export_to_fd/ibv_context_to_fd/ibv_import_pd, see
https://marc.info/?l=linux-rdma&m=156155921212805&w=2, but having just one
optimized smbdirect implementation on Linux (for userspace and kernel consumers)
still seems to be the better solution. Samba needs to serve files from the
kernel space and a kernel driver will very likely allows less memory copies of the
file data.

For now we use /dev/smbdirect and custom ioctl() calls instead of adding new
syscalls.

1.2.1 Implementation in the Kernel (basics)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

There's a prototype that is mostly functional, but it lacks a lot of error
checks. It is full of memory leaks and some features are not yet implemented
completely or only in a sync fashion.

Complete the existing prototype with correct error checking.
Make sure the Windows SMB-Direct test suite passes against Samba.

Add useful trace points in order to debug the driver and find possible performance bottlenecks.

Use a ring buffer interface similar to the io_uring (See https://lwn.net/Articles/778411/). Maybe add something like an async ioctl to the uring interface or true async sendmsg(MSG_OOB).

Experiment/research regarding usage of (transparent) huge pages and ways to use just one memory registration for an 8MByte buffer.

We need to batch ib_post_send() requests and only ask for completions on the last send that belongs to a higher level operation.

ib_dma_map_sg() merges elements and we might be able to reduce the size of the smbdirect_buffer_descriptors_v1 array. Each element can hold max_frmr_depth sgl entries. Using just one allows the SEND_WITH_INV to work effectively.

Add some restrictions with io_account_mem() and unprivileged port numbers.

=> 3-4w

1.2.2 Userspace API for smbdirect:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The simplified API used in the current prototype looks like this:

```c
int smbdirect_socket(int family, int type, int protocol);
int smbdirect_connection_get_parameters(int sockfd,
    struct smbdirect_connection_parameters *params);
ssize_t smbdirect_rdma_v1_register(int sockfd,
    struct smbdirect_buffer_descriptors_v1 *local,
    int iovcnt, const struct iovec *iov);
ssize_t smbdirect_rdma_v1_deregister(int sockfd,
    const struct smbdirect_buffer_descriptors_v1 *local);
ssize_t smbdirect_rdma_v1_writev(int sockfd,
    const struct smbdirect_buffer_descriptors_v1 *remote,
    int iovcnt, const struct iovec *iov);
ssize_t smbdirect_rdma_v1_readv(int sockfd,
    const struct smbdirect_buffer_descriptors_v1 *remote,
    int iovcnt, const struct iovec *iov);
void smbdirect_buffer_invalidate_cmsg_prepare(struct msghdr *msg,
    uint8_t *buf,
    size_t buflen,
    const struct smbdirect_buffer_descriptor_v1 *desc);
```

The created socket provides a stream with 4 byte (big endian) length delimiters, in order to provide a compatible with SMB over TCP.

An optimized version with sendfile/recvfile performance like this:

```c
ssize_t smbdirect_rdma_v1_write_from_file(int sockfd,
    const struct smbdirect_buffer_descriptors_v1 *remote,
    int file_fd,
    off_t file_offset,
    size_t length);
ssize_t smbdirect_rdma_v1_read_to_file(int sockfd,
    const struct smbdirect_buffer_descriptors_v1 *remote,
    int file_fd,
    off_t file_offset,
    size_t length);
```

The smbdirect_buffer_descriptors_v1 based APIs should be converted to mmap’ed ring buffer interface, similar to io_uring. At least the write/read calls need to be async, the above prototypes are only simplified examples.

These would be implemented using in kernel or userspace bounce buffers in the first version, later we may be able to use the page cache directly or even rdma directly from the block device.

We also need ways to specify parameters like timeouts, max credits, max io size and more before connect/listen. So these can be specified per socket from
userspace, as it makes it easier to find the optimized values.
The following is required for smbd to announce RDMA interfaces.

```c
int smbdirect_netif_rdma_capable(const char *ifname);
```

1.2.3 Kernelspace API for smbdirect
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The simplified API used in the current prototype looks like this:

```c
int smbdirect_sock_create_kern(struct net *net, int family, int type, int protocol, struct socket **res);
int smbdirect_kern_connection_get_parameters(struct socket *sock, struct smbdirect_connection_parameters *params);
ssize_t smbdirect_kern_rdma_v1_register_pages(struct socket *sock, struct smbdirect_buffer_descriptors_v1 *local, struct page *pages[], int num_pages, int pagesz, int fp_ofs, int lp_len);
ssize_t smbdirect_kern_rdma_v1_deregister(struct socket *sock, struct smbdirect_buffer_descriptors_v1 *local);
```

Most likely we can keep this api, if really required
smbdirect_kern_rdma_v1_deregister can be made async.

Convert fs/cifs/ to use this api as alternative to fs/cifs/smbdirect.c and compare the performance results.

This is not strictly required, but it will help in order to find bugs and compare the functionality and performance with the existing fs/cifs/smbdirect.c implementation of SMB-Direct. It will also help to get the code merged into the mainline kernel.

1.2.4 Write a standalone testsuite
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

This should not use SMB3, but a small custom echo-like protocol that is able to test the api and protocol.

The protocol would have opcodes like this:

```c
REQ_HEADER {
    __le32 message_id;
};
REP_HEADER {
    __le32 message_id;
    __le32 status;
};
DESCRIPTOR {
    __le64 iova;
    __le32 rkey;
    __le32 length;
};
ECHO-REQ(REQ_HEADER hdr, __le16 echo_factor, __le32 data_len, __le32 max_response, u8 data[data_len])
ECHO-REP(REP_HEADER hdr, __le32 data_len, u8 data[data_len])
/*
 * SETUP-LBUF creates a buffer on the
 * server where the content will be
 * for (ofs=0; ofs < size; ofs += 8) {
 *    PUSH_UINT64_LE(buf, ofs, nonce ^ ofs);
 * }
```
This simulates a file, the server maintains an array of LBUF (files), lbuf_idx is the index into that array.

```
SETUP-LBUF-REQ(REQ_HEADER hdr, u32 lbuf_idx, u32 size, u64 nonce);
SETUP-LBUF-REP(REP_HEADER hdr)
READ-LBUF-TO-DESCS-REQ(REQ_HEADER hdr, u32 lbuf_idx, u32 flags, /* SEND_WITH_INV */ u32 offset, u32 length, u32 num_descs, DESCRIPOR descs[])
READ-LBUF-TO-DESCS-REP(REP_HEADER hdr, u32 nread)
WRITE-DESCS-TO-LBUF-REQ(REQ_HEADER hdr, u32 lbuf_idx, u32 flags, /* SEND_WITH_INV */ u32 offset, u32 length, u32 num_descs, DESCRIPOR descs[])
WRITE-DESCS-TO-LBUF-REP(REP_HEADER hdr, u32 nwritten)
```

It can be used as a regression and performance test for the kernel parts without having to deal with Samba or the fs/cifs/ code.

```c++
1w
```

```c++
6-7w (~1.5-1.75M) (1.2 Linux Kernel Driver)
```

1.3 Add support for SMB-Direct to Samba

```
1.3.1 Add support to the generic client library
```

There's already a prototype to let 'smbclient' use SMB-Direct with RDMA. But it's not used automatically as multi-channel support with interface detection is missing.

For a start we need to be able to write tests and allow smbcnt to specific the transport protocol e.g. NetBios, TCP, SMB-Direct (or later QUIC).

This is required in order to test without relying just on highlevel tests with Windows.

```c++
We also need to write some smbtorture tests.
```

```c++
1-2w
```

1.3.2 Add support to smbd READ/WRITE

```
There's already a prototype that allows connections via SMB-Direct including RDMA read and writes, but they use a sync interface in order to do the RDMA read/write operation these need to make async.

We should maybe able to implement some optimization like sendfile() in order to avoid data copies.

A custom ioctl on the SMB-Direct socket or some interaction with io_uring and linked operations are possible solutions.
```
Can we do RDMA read/writes from/to the page cache of files?

1.3.3 Plug SMB-Direct into the interface detection

We need a way to specify if smbd should listen on SMB-Direct sockets. If so we should automatically mark interfaces exported by the Multi-Channel interface detection as RDMA capable.

1w

3-4w (~0.75-1M) (1.3 Add support for SMB-Direct to Samba)

10-14w (~2.5-3.5M) (1 Tasks required to add SMB 3 SMB-Direct support)

2. Tasks required to bring SMB 3 SMB-Direct to Upstream

2.1 Upstream Samba Changes

2.1.1 Automated Multi-Channel Testing

We need to have regression tests in our autobuild/gitlab-ci in order to remove the "experimental" flags from the "server multi channel support" option and enable it by default.

Note that Multi-Channel is not a Linux-specific feature, which means we can't rely on Linux containers/namespaces.

scope:

Socket wrapper maintains a socket_info structure for each tcp/udp socket. As it's possible to have more than one file descriptor for a socket, there can be multiple socket_info_fd structures pointing to a single socket_info structure. Currently socket wrapper maintains a global linked list, without any thread safety.

Socket wrapper doesn't support fd-passing of tcp/udp sockets, the fd is passed, but it appears as unix domain socket in the other process.

tasks:

- We need to maintain a small file using mmap and protected by pthread robust mutexes. E.g. one file per local ip address.
- The path specified in SOCKET_WRAPPER_FD_PASSING_DB will be used as the file name, if this is not specified we'll use malloc'ed and fd-passing is not enabled.
- The file contains a header (with magic, unique id, size and free-list pointer) followed by an array of socket_info structures.
- The socket_info_fd structures will only maintain the index into the mmap'ed array.
- fd-passing is limited to fixed number (~ 127), this should be more than enough for typical caller (Samba would just use 1).
- In order to do fd-passing of tcp/udp sockets, we'll create a pipe (or similar) where we write an array of with indexes into the mmap'ed array into the write end of the pipe. We would also pass the device/inode and a unique identifier for the file.
- The read end of the pipe is then passed as the last fd to the destination process. The destination process can rebuild the socket_info_fd structures by reading the array indexes out of the read end of the pipe.
- A tricky part will be the reference counting in the database entries. The sender needs to write the data into the pipe and increment the refcounts in the db file before calling sendmsg(). The sender may hold a mutex for each socket during sendmsg().
- In order to allow multiple threads (or processes) to share a single socket we need to add mutex protection in quite a few places. In the most common cases there won't be any contention on the mutexes, but it will guarantee correctness for the corner cases which happens for fd-passing.
- See https://gitlab.com/anoopcs/socket_wrapper.git
  gitlab-anoopcs-socket_wrapper/fd-passing-final
- See https://gitlab.com/metze/socket_wrapper.git
  gitlab-metze-socket_wrapper/fd-passing
- TESTS
  ==> 2-3w

2.1.2 Automated SMB-Direct Testing
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Samba requires features to be tested by every push to the upstream repositories.

There is support for using network namespaces (on Linux) instead of socket wrapper for testing.

Maybe it's possible to integrate something like this into the gitlab-ci. We could download the kernel driver sources and build/load the kernel module before starting the SMB-Direct specific tests inside a KVM machine (it might not be possible to do that from within a docker container).

Or we just have a stable version of the kernel driver installed on a private runner.

  ==> 2-3w

  ==> 4-6w (~1-1.5M) (2.1 Upstream Samba Changes)

2.2 Linux Kernel Driver
------------------------

2.2.1 Upstream integration
~~~~~~~~~~~~~~~~~~~~~~~~~~

Make use of the correct kernel APIs.

The initial design is based on /dev/smbdirect and ioctl() calls to create and operate on a socket.

In order to bring this code upstream it's very likely that we have to adopt to modern kernel APIs.

It's very likely to take lot of time to discuss with the kernel community about the correct ways to integrate the driver into the upstream kernel.

We may also have to rewrite parts of the driver in order to make it acceptable.

It might be easier to propose only the in kernel api first and let it replace fs/cifs/smbdirect.* to be used in fs/cifs. Small selfcontained chunks are much easier to review and have a much higher chance to get accepted.

Here is just a list of things which are likely to be considered/researched/discussed:
- Should we introduce a PF_SMBDIRECT to replace the ioctl() on /dev/smbdirect in order to allow the socket() syscall to create the socket?
- Should we use getsockopt/setsockopt to replace all/some custom ioctl() calls?
- Should we try to integrate our custom async handling with io_uring?
- Do we need special handling in order to interact with network namespaces
  (See register_pernet_subsys())
  (See "rdma sys set netns shared"?)
- Do we need to interact with netlink sockets?
- Should we use common structures for buffer memory handling
  See sk_buff/sock_kmalloc/sk_receive_queue/sk_wmem_alloc/sk_memcg/
  sk_stream_wait_memory.sk_enter_memroy_pressure.sk_prot_mem_limits/
- Can we use sock_def_write_space instead of our own?
- Do we need to implement any/more of common getsockopt/setsockopt
  opcodes?
- Should be make use of sk_stream_error(), see sk_err/sk_err_soft
  and SIGPIPE
- Do want to interact nicely with common tools like
  "netstat" or "ss" and replace the custom diagnostics from /proc/smbdirect?
  See also sock_diag_register/sock_diag_handler.
- Do we need to implement some of the SK_FLAGS_TIMESTAMP logic?
- We may need to interact with various security/filter layers
  in the kernel. Research on security_socket_post_create,
  BPF_CGROUP_PRE_CONNECT_ENABLED, netfilter, BFP filters
  and related things.

While doing the integration work we may find bugs or limitations in
existing kernel infrastructure, we may need to do some generic cleanups in
all sorts of kernel subsystems.

As smbdirect should work with every RDMA hardware or software driver in
the kernel, we may need to analyse problems related hardware or the
related driver. We may hit unsolveable problems, but in that case we
should at least try to fail gracefully (at best before establishing a
connection). E.g. we rely on FRWR (Fast Registration Work Requests)
support and check for IB_DEVICE_MEM_MGT_EXTENSIONS.

==> 2−4w

==> 2−4w (~0.5−1M) (2.2 Linux Kernel Driver)

==> 6−10w (~1.5−2.5M) (2 Tasks required to bring SMB 3 SMB−Direct to Upstream)

c) Development Requirements
===========================

As the success of implementing SMB−Direct (unlike other typical SMB
features) is bound to the available Hardware and also the Linux Kernel
(including the device specific driver), we need to clarify the project
boundaries in order to build a common expectation of project's outcome.

As each vendor will most likely have a specific hardware setup that
should work in the end, we propose that each vendor provides a hardware
setup that can be used during the development.

The requirements for each setup are these:

5 x Hardware Computers with:
- remote KVM-over-IP access in order to control the
  BIOS/boot sequence and allow forced hardware resets.
- 2 should be servers, similar to the product the
  vendor will later ship, they might have fast
  disks (ssd/nvme)
- 2 can be typical client machines
- 1 will be used to compile and monitor
- 1 server and 1 client and the monitor host
  will run with Linux
  (Ubuntu-18.04 (or newer) amd64 at least in the beginning,
  maybe with dual boot to the vendor's favorit distribution)
- 1 server and 1 client will run with Windows
  (I'd use Windows Server 2019 Datacenter on both,
  maybe dual boot to 2016 and/or 2012R2)
- All of them are reachable via ssh or rdp on
  a normal (at least 1 Gbit) network interface from
  the outside.
- The Windows server and client will be taken
  as a reference in order to get a feeling for
  what performance could look like.
- For each R−NIC flavor we need at one card in each
  computer. All 5 cards need to be the exact same model
with the same firmware version. If more than flavor
is desired we can have more than one card in each computer.
- Most R-NICs have more than one port, we should use at least
  2 ports. Port 1 of all cards should be attached to the
  same switch, that same applies to port 2 and others.
  If a switch is able to handle 10 ports it should be
  ok to connect all 10 (2x5) ports to the same switch.
- If the interaction with ctdb should be implemented/tested,
  we may need more hardware.

All switches need to offer a monitoring/mirroring port where the
monitoring host can capture traffic between the other hosts, this is a
very important requirement.

We'll start with DiskSpd.exe based tests from the Windows Client, see how

If the vendor has a special workload in mind the software needs to be
installed on a client (most likely the Windows client) and run against the
Windows server as reference.

While we take Windows as a reference it's very likely its performance
won't be reached (at least initially, but maybe forever). But the goal is
that the SMB-Direct code path is at least not slower or more cpu consuming
as the TCP code path for the typical DiskSpd.exe workloads.

The current linux kernel module prototype compiles against 4.10 and higher
versions (currently v5.5). Most of the testing was done with the 5.3
kernel from ubuntu 18.04.

The new io_uring feature of 5.1 (and higher kernels) provides async io
with reduced context switches, we may rely on this or at least use some
concepts. See https://lwn.net/Articles/778411/

As it's the goal to bring the kernel driver upstream (in the long run),
we may only support recent kernel versions (at the time we finalize the
project). Backports to older kernel versions are not part of the project.

As it's not trivial to get kernel changes upstream, we may have to
maintain the out of tree module for a while. We had some discussions with
the maintainer of the cifs.ko kernel module and it might be possible to
getter the kernel changes merged step by step. A first part could replace
the existing SMB-Direct client implementation in the smb3 kernel client.

For the Samba part we'll develop against the current master branch. The
Samba-Team has high standards for getting features pushed to the upstream
master tree (which is the base for future releases). Typically new
features need regression tests. Developing/finishing such tests and make
them part of the pre-push autobuild/gitlab-ci requires a lot of work
(maybe more then developing the features itself!). This means it's likely
to maintain the patches out of the master tree for a while. Integration of
the tests into the ci is not part of the project.

The maintenance of the out of tree branches (after the main project
finished) is not part of the project.